Graph Reconstruction with a Betweenness Oracle

Mikkel Abrahamsen*[1], Greg Bodwin[2], Eva Rotenberg[3], and Morten Stöckel†[4]

1 **University of Copenhagen Department of Computer Science, Denmark**
`roden@di.ku.dk`
2 **Stanford University, Department of Computer Science, USA**
`gbodwin@cs.stanford.edu`
3 **University of Copenhagen Department of Computer Science, Denmark**
`miab@di.ku.dk`
4 **University of Copenhagen Department of Computer Science, Denmark**
`most@di.ku.dk`

Abstract

Graph reconstruction algorithms seek to learn a hidden graph by repeatedly querying a black-box oracle for information about the graph structure. Perhaps the most well studied and applied version of the problem uses a distance oracle, which can report the shortest path distance between any pair of nodes.

We introduce and study the *betweenness* oracle, where $\mathrm{bet}(a, m, z)$ is true iff m lies on a shortest path between a and z. This oracle is strictly weaker than a distance oracle, in the sense that a betweenness query can be simulated by a constant number of distance queries, but not vice versa. Despite this, we are able to develop betweenness reconstruction algorithms that match the current state of the art for distance reconstruction, and even improve it for certain types of graphs. We obtain the following algorithms:

1. Reconstruction of general graphs in $O(n^2)$ queries
2. Reconstruction of degree-bounded graphs in $\tilde{O}(n^{3/2})$ queries
3. Reconstruction of geodetic degree-bounded graphs in $\tilde{O}(n)$ queries

In addition to being a fundamental graph theoretic problem with some natural applications, our new results shed light on some avenues for progress in the distance reconstruction problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph reconstruction, bounded degree graphs, query complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.5

1 Introduction

Background and Applications

A major subfield of graph algorithms is that of *graph reconstruction*, in which one must determine the edges of a hidden graph using a black-box oracle that reveals a certain type of information about the graph. This model is typically used to study systems in which it is costly or slow to make measurements on the graph; the subject of reconstruction research is therefore to find strategies that learn the graph with low worst-case *query complexity*. The

* Research partially supported by Mikkel Thorup's Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career programme.
† Research supported by Villum Fonden.

offline computation time of these algorithms is not a central point of concern, although it is generally expected to be polynomial.

This framework captures a number of important problems in computational biology. In one well-studied example, researchers wish to learn evolutionary trees, but only have tools to query for the distance in the unknown tree between an arbitrary pair of species (see [25, 13, 16, 22] for work on this problem). Each of these queries requires some research effort, and so one hopes to choose queries efficiently so as to maximize the information gained from each one. Another reconstruction problem is implicitly studied in genome sequencing; in this version, the underlying oracle takes a node subset S and reports whether or not this set is independent (see [1, 2, 4, 3, 11] for work on this problem). Yet another version is central to bioinformatics, in which the underlying oracle must count the number of internal edges in a node subset S (see [6, 12] for work on this problem). There is also the *network reconstruction problem*, in which one runs tests on a system to learn the topology of a decentralized network – for example, one might hope to discover the graph of the internet by querying for connectivity information between routers (see [5, 8, 9, 10] for work on this problem). Reconstruction problems also appear in network tomography [7, 23], probability theory [19], and other fields. In parallel, there has been work on the closely related *graph verification problem* in which one must simply confirm that the oracle matches a graph taken on input. For some examples of work on this problem, see [15, 5, 7, 10].

We introduce a new oracle in this paper, which is most similar to the distance oracle. We will therefore proceed through this introduction with our attention restricted to the distance reconstruction problem. If the reader is interested in more exposition on reconstruction/verification under the *other* oracles, we refer them to [15, 21, 3, 5, 1, 2, 4, 18] and the citations therein.

Our New Oracle

We introduce and study the *betweenness* oracle, which is defined such that $\mathrm{bet}(a, m, z)$ is true iff there is a shortest path between a and z that contains m.

There are a few reasons why we consider this oracle worthy of study. The first is that this oracle generalizes the distance oracle due to the following equivalence:

$$\mathrm{bet}(a, m, z) \qquad \Leftrightarrow \qquad \mathrm{dist}(a, m) + \mathrm{dist}(m, z) = \mathrm{dist}(a, z)$$

It is therefore possible to simulate a betweenness query with a constant number of distance queries, and so any query complexity obtained for the betweenness reconstruction problem is automatically achieved for the distance reconstruction problem as well. It is not hard to see that one *cannot* in general simulate a distance query with a constant number of betweenness queries: in fact, given no other information about the graph, at least $\Omega(n)$ betweenness queries are required to learn a single pairwise distance. In this sense, we regard the betweenness oracle as *much weaker* than the distance oracle.

Given this intuition, it is very natural to expect that betweenness reconstruction would require a higher query complexity than distance reconstruction. However, as of our new results in this paper, this is not the case! We are able to match the most important state of the art results for distance reconstruction, and even improve them for certain classes of hidden graphs.

This surprise sheds some light on the central open problem [17, 15] of distance reconstruction, which is to obtain an algorithm with query complexity $n^{1+o(1)}$ for degree-bounded graphs. We are able to match the current best query complexity ($\tilde{O}(n^{3/2})$ from [17]) for this problem using only a betweenness oracle. This suggests that the current distance reconstruction algorithms are unable to exploit the additional power of their oracle in a meaningful way, and that further progress can perhaps be made through using this information in a more careful manner.

Our second reason for studying this oracle is purely practical: it is useful to study reconstruction problems in a context free of a distance model. For example, in the evolutionary tree literature, the distance reconstruction approach asks biologists to devise a model of evolutionary distance, and then use this to reconstruct the order of speciation events [13]. There is no single definitive method for modeling evolutionary distance, and all popular methods for measuring these distances are somewhat error-prone and require certain underlying assumptions to be made (see [20] for a discussion of evolutionary distance estimation). In contrast, the betweenness reconstruction approach allows one to reconstruct the tree without having to worry about a model of evolutionary distance. Instead, one only needs to answer yes-or-no betweenness queries about the tree structure. Per the distance/betweenness relationship given above, this is a strictly easier task.

Our third reason for valuing the betweenness oracle is purely theoretical: intuitively, we consider the betweenness oracle to be by far the weakest oracle under which nontrivial reconstruction results have been obtained. Previously, this distinction belonged to the distance oracle. All other oracles receiving significant attention had non-constant arity and/or returned a polynomial number of bits on output, and the best reconstruction results for these other oracles had significantly lower query complexity (typically $\tilde{O}(n)$) than was known for distance oracles ($O(n^2)$ for general graphs); see [21] for a survey. Since our betweenness oracle is strictly weaker than a distance oracle, it should be regarded as the new weakest studied oracle.

Our Results

Our first main result is:

▶ **Theorem.** *There is a betweenness reconstruction algorithm with a query complexity of $O(n^2)$ (this algorithm makes no assumptions about the hidden graph).*

For general graphs, the trivial brute force distance reconstruction algorithm uses $O(n^2)$ queries, but the brute force betweenness reconstruction algorithm uses $O(n^3)$ queries, so some cleverness is required to avoid making all possible queries. An interesting consequence of our new oracle is apparent here: because our oracle returns only a single bit of information, there is a simple matching information-theoretic lower bound matching this upper bound. A graph encodes $\Theta(n^2)$ bits of information, each betweenness query reports a single bit of information, and so at least $\Omega(n^2)$ queries are required in the worst case.

This particular argument fails for distance reconstruction, which reports $\log n$ bits of information per query. However, a matching lower bound is still known. A simple lower bound is a graph with $n - 2$ isolated vertices plus a pair of vertices connected by a single edge; the only possible reconstruction algorithm here is a brute-force search for the missing edge. To forbid this construction, it is common to assume that the hidden graph is connected. However, this is still not enough: Reyzin & Srivastava [21] have exhibited a constant-depth tree that requires $\Omega(n^2)$ distance queries to reconstruct. This tree has a root of degree $\Omega(n)$; therefore, the most natural way to forbid this construction is to parametrize the solution quality on the maximum degree Δ of the graph. In many natural applications Δ is constant or at most $n^{o(1)}$ [17, 13], so in this parametrization it is considered much more important to reduce the dependence on n than to reduce the dependence on Δ.

Our next main result fits within this parametrization. We prove:

▶ **Theorem.** *When the hidden graph is connected with maximum degree Δ, there is a betweenness reconstruction algorithm with a query complexity of $\tilde{O}(n^{3/2} \cdot \Delta^4)$.*

This matches the query complexity obtained by Mathieu & Zhou [17] for distance reconstruction. It is generally considered to be the foremost open problem [17, 15] to obtain distance reconstruction for degree-bounded graphs with a query complexity of $n^{1+o(1)} \cdot f(\Delta)$ for any function f. We consider it quite interesting that the state-of-the-art against this open problem can be achieved by our betweenness oracle: it suggests that the current algorithms for this problem do not integrally exploit the fact that they receive full distance information, and that an avenue for progress is to devise an algorithm that uses distance information in a new way.

While this $n^{3/2}$ bound remains unbroken, the problem has been solved for certain specific types of graphs. In particular, degree-bounded distance reconstruction algorithms with query complexities of $\tilde{O}(n \cdot f(\Delta))$ are known when the hidden graph is outerplanar [17] or chordal [15]. Our third main result is in this regime. We prove:

▶ **Theorem.** *When the hidden graph is connected, geodetic (i.e., there is a unique shortest path between every pair of nodes), and has maximum degree Δ, there is a betweenness reconstruction algorithm with a query complexity of $\tilde{O}\left(n \cdot \Delta^3\right)$.*

Geodetic graphs can be seen as a generalization of trees; in this vein, our upper bound comes very close to the distance reconstruction lower bound of $\tilde{\Omega}(n \cdot \Delta)$ for trees given in [16]. Due to the relationship between betweenness and distance queries discussed above, an identical upper bound is immediate for distance reconstruction, thereby improving the distance query complexity for these graphs by a factor of $O(\sqrt{n})$ over [17].

Finally, we remark that our algorithms in the two latter cases are Monte Carlo, and that the error probability can be tuned to an arbitrarily small inverse polynomial in n by increasing the hidden constant.

2 Terminology

We will first collect some previously used notations and definitions from the world of graph theory that will be used repeatedly throughout the paper.

We reserve $G = (V, E)$ to refer to the hidden graph being reconstructed, and $|V| = n$. Unless otherwise noted, the graph is undirected and unweighted. Given a node subset $U \subset V$, $G[U]$ is the subgraph induced by U. The neighborhood of a node v (i.e. all nodes u such that $(v, u) \in E$) is denoted by $N(v)$. The neighborhood of a node set S (i.e. the union of $N(s) - S$ for all $s \in S$) is denoted by $N(S)$.

▶ **Definition 1** (Starshaped). A node subset $X \subset V$ is *starshaped* with respect to a *center* $x \in X$ if, for all $v \in X$, every shortest path from x to v is entirely contained in X.

Note that the same subset may be starshaped with respect to several centers.

▶ **Definition 2** (Layer Structure). Given a starshaped set $X \subset V$ with center x, a node $v \in X$ is said to be in layer i of X if $\operatorname{dist}(x, v) = i$. The set of nodes in layer i is denoted L_i^X. The X superscript is suppressed when clear from context. (See Figure 1.)

▶ **Definition 3** (Spanning Tree). Given a starshaped set $X \subset V$ with center x, a subgraph $\mathcal{T}_X$ of $G[X]$ is a *spanning tree* of X if it is a tree with the property that for all $v \in X$, $\mathcal{T}_X$ includes a shortest path in X from x to v.

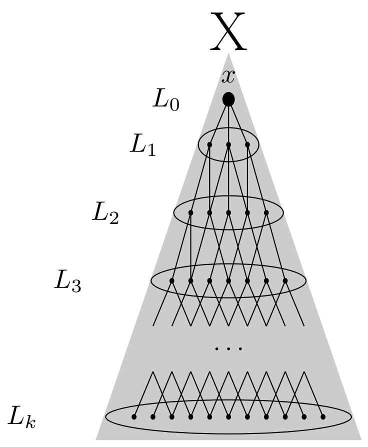

Figure 1 The layer structure for a starshaped set X with center x, as well as the shortest path graph.

▶ **Definition 4** (Shortest Path Graph). Given a starshaped set $X \subset V$ with center x, the *shortest path graph* of X is the subgraph $\mathcal{S}_X$ of $G[X]$ defined by removing all edges (a, b) where a and b are in the same layer of X.

By orienting all edges of the Shortest Path Graph away from x, one obtains a DAG with source x, and thus, a notion of ancestry and parenthood between the vertices of X. We formalize in the following way:

▶ **Definition 5** (Tree Definitions). Let X be a starshaped set with center x. If $v \in L_i^X$, then u is a *parent* of v if $u \in N(v) \cap L_{i-1}^X$, or u is a *child* of v if $u \in N(v) \cap L_{i+1}^X$. The *ancestor* relation is the transitive closure of the parent relation, and the *descendant* relation is the transitive closure of the child relation. By convention, a node is both its own ancestor and its own descendant. The set of descendants of a node v is denoted $\mathcal{D}(v)$, and the set of ancestors of a node v is denoted $\mathcal{A}(v)$. A node is a *leaf* if it has no children.

▶ **Definition 6** (Geodetic Graphs). A graph is *geodetic* if, for each node pair, there is a unique shortest path between these nodes.

Note that, for a geodetic graph, there is a unique spanning tree of any starshaped set X, and this spanning tree is also the shortest path graph.

▶ **Definition 7** (Betweenness). We define the relation $\operatorname{bet}(\cdot, \cdot, \cdot)$ such that $\operatorname{bet}(a, m, z)$ is true iff there exists a shortest path in G between a and z that includes m.

▶ **Definition 8** (Ancestry). For a "root node" $r \in V$, we define the relation $\operatorname{anc}_r(\cdot, \cdot)$ such that $\operatorname{anc}_r(u, v) \leftrightarrow \operatorname{bet}(r, u, v)$. We will sometimes use this query in place of a bet query to emphasize that our goal is to test membership of u in $\mathcal{A}(v)$.

3 Reconstruction of General Graphs

There are $\Omega(n^3)$ different betweenness queries that one can ask on any given graph. However, we prove:

▶ **Theorem 9.** *One can learn the edges of a hidden graph using $O(n^2)$ betweenness queries (with no assumptions made about the hidden graph; G can be disconnected and have any maximum degree).*

Note that there is a simple matching information-theoretic lower bound for general graphs: a graph encodes $\Omega(n^2)$ bits of information, and since each betweenness query returns a single bit of information, at least this many queries are required to learn the graph.

The upper bound proceeds in two steps. The first is to discover the connected components of the graph:

▶ **Claim 10.** *One can discover the connected components of a hidden graph using $O(n^2)$ betweenness queries.*

Proof. For all $u, v \in V$, query $\text{bet}(u, u, v)$. The query will be true iff there exists a shortest path between u and v, which holds only if u and v belong to the same connected component of the graph. ◀

The second step in the upper bound is to discover the edges of a single connected component. We will prove this result at a higher level of generality, as the techniques will be useful again later in the paper.

▶ **Claim 11.** *Given a starshaped set X with center x, as well as the shortest path graph of X, one can decide whether or not there exists an edge between any two nodes u, v in the hidden graph using $O(1)$ betweenness queries.*

Proof. If u, v belong to different layers, then the task is trivial: if there is an edge between them, then it is in the shortest path graph, so one must simply consult the shortest path graph and no queries need to be executed. So assume u, v are in the same layer. Let node a be a least common ancestor of u and v; i.e. choose a node $a \in \mathcal{A}(u) \cap \mathcal{A}(v)$ with the highest level. Since u, v are in the same layer, we then have $\text{dist}(u, a) = \text{dist}(v, a) =: h$. Let b be a child of a along a shortest path between a and u. Then $\text{dist}(b, u) = h - 1$ and $\text{dist}(b, v) \in \{h, h+1\}$. Query $\text{bet}(v, a, b)$. If this query is true, then we have $\text{dist}(b, v) = h+1$, which then implies that the edge (u, v) does *not* exist in the hidden graph. If this query is false, then we have $\text{dist}(b, v) = h$. We then query $\text{bet}(v, u, b)$, which will be true iff the edge (u, v) exists in the hidden graph. ◀

▶ **Claim 12.** *Let $X \subset V$ be a starshaped set with center x. One can discover all edges in $G[X]$ in $O(|X|^2)$ betweenness queries.*

Note that if X is an entire connected component of the graph, then it is trivially starshaped with respect to any $x \in X$, and so along with Claim 10, this result implies Theorem 9.

Proof. The algorithm proceeds in two steps. First, we query $\text{anc}_x(u, v)$ for all $u, v \in X$, and we claim that this information is sufficient to learn the shortest path graph $\mathcal{S}_X$. We can learn the layers by the following recursive formula:

$$L_0 = \{x\},$$
$$L_i = \left\{ v \in (X - \bigcup_{j<i} L_j) \quad | \quad (\mathcal{A}(v) - \{v\}) \subset \bigcup_{j<i} L_j \right\}.$$

This formula states that if layers $\{0, \ldots, i-1\}$ are known, then the i^{th} layer can be determined as the set of nodes v such that all ancestors (besides the node itself) belong to these first layers. Once the layers are known, for any node $v \in L_i$, we can determine its neighbors in L_{i-1} as the set

$$N(v) \cap L_{i-1} = \mathcal{A}(v) \cap L_{i-1}.$$

Applied to all $v \in X$, this is sufficient to learn every edge in $\mathcal{S}_X$. We can now use Claim 11 on all pairs of edges in the graph to detect the existence/nonexistence of every possible edge in $O(|X|^2)$ queries. ◀

4 Finding Splitting Nodes

For the rest of this paper, we will assume that G is connected with maximum degree Δ.

▶ **Lemma 13.** *Every starshaped set X with center x has a node $s \in X$ with the property*

$$\left\lceil \frac{|X|}{3\Delta} \right\rceil \leq |\mathcal{D}(s)| \leq \left\lceil \frac{|X|}{3} \right\rceil .$$

Proof. Let ℓ be a leaf of X. We then have

$$|\mathcal{D}(\ell)| = |\{\ell\}| = 1.$$

And so if $|X| \leq 3\Delta$, then ℓ satisfies the property. Otherwise, assume $|X| > 3\Delta$.

We will now proceed with an intermediate value type argument. Initialize a node $p \leftarrow x$, and then repeatedly find the child $c_{\max}$ of p with the greatest number of descendants and set $p \leftarrow c_{\max}$. Stop this process once p is set to a leaf. At each step, we have

$$\left| \bigcup_{c \text{ is a child of } p} \mathcal{D}(c) \right| = |\mathcal{D}(p) - \{p\}| = |\mathcal{D}(p)| - 1.$$

Since p has at most Δ children, the union is taken over Δ elements, and so by a (inverse) union bound, the average child has at least $(|\mathcal{D}(p)| - 1)/\Delta$ descendants, and so $c_{\max}$ has at least this many descendants. So if

$$|\mathcal{D}(p)| > \left\lceil \frac{|X|}{3} \right\rceil$$

then

$$|\mathcal{D}(c_{\max})| \geq \left\lceil \left\lceil \frac{|X|}{3} \right\rceil / \Delta \right\rceil \geq \left\lceil \frac{|X|}{3\Delta} \right\rceil .$$

Note that the size of the set $\{v \in X \mid \mathrm{anc}_x(p, v)\}$ is strictly decreasing in each step. Its initial value is $|X|$ and its final value is 1. Therefore, at some point during this process, we have

$$|\mathcal{D}(p)| > \left\lceil \frac{|X|}{3} \right\rceil \quad \text{and} \quad |\mathcal{D}(c_{\max})| \leq \left\lceil \frac{|X|}{3} \right\rceil .$$

From the above argument, we also have

$$|\mathcal{D}(c_{\max})| \geq \left\lceil \frac{|X|}{3\Delta} \right\rceil .$$

and so $c_{\max}$ satisfies our property. ◀

A node is a *splitting node* if it satisfies a relaxed version of this condition (see Figure 2).

▶ **Definition 14.** A node s satisfying

$$\left\lceil \frac{|X|}{4\Delta} \right\rceil \leq |\mathcal{D}(s)| \leq \left\lceil \frac{|X|}{2} \right\rceil$$

is called a *splitting node*.

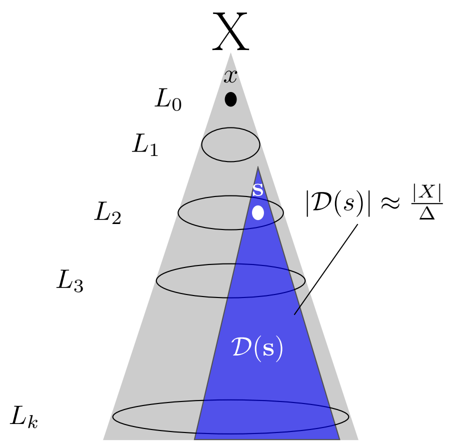

Figure 2 A representation of a splitting node s of the starshaped set X.

▶ **Lemma 15.** *Given a starshaped set X with center x with hidden edges, there is a randomized algorithm that uses $\tilde{O}(|X| \cdot \Delta)$ betweenness queries*[1] *and, with high probability, finds a splitting node $s \in X$.*

Proof. Iterate over each node $v \in X$. Let R be a uniform random sample S of $k \cdot \log |X| \cdot \log\log |X| \cdot \Delta$ nodes, where k is a constant whose size determines the probability of success. We then query $\mathrm{anc}_x(v, r)$ for each $r \in R$, and we estimate $|\mathcal{D}(v)|$ as:

$$\widehat{|\mathcal{D}(v)|} = |\mathcal{D}(v) \cap R| \cdot \frac{|X|}{|R|}$$

By standard Chernoff bounds, for sufficiently large k, we have

$$|\mathcal{D}(v)| - \frac{|X|}{29\Delta} \leq \widehat{|\mathcal{D}(v)|} \leq |\mathcal{D}(v)| + \frac{|X|}{29\Delta}$$

with high probability. We then accept v as a splitting node iff

$$\left\lceil \frac{|X|}{3.5\Delta} \right\rceil \leq \widehat{|\mathcal{D}(v)|} \leq \left\lceil \frac{|X|}{2.5} \right\rceil .$$

Then, for an accepted node v, if $|X|$ is sufficiently large then the following inequalities hold with high probability:

$$\left\lceil \frac{|X|}{4\Delta} \right\rceil < \left\lceil \frac{|X|}{3.5\Delta} \right\rceil - \frac{|X|}{29\Delta} \leq |\mathcal{D}(v)| \leq \left\lceil \frac{|X|}{2.5} \right\rceil + \frac{|X|}{29\Delta} < \left\lceil \frac{|X|}{2} \right\rceil ,$$

and hence v is a splitting node with high probability. Similarly, we see that a non-splitting node is rejected with high probability. Additionally, with high probability, we will accept any node satisfying the condition in Lemma 13. Since at least one such node exists, this process will find a splitting node with high probability.

We use $O(\log |X| \cdot \log\log |X| \cdot \Delta)$ queries per node in X, so the total number of queries used by this process is $O(|X| \cdot \log |X| \cdot \log\log |X| \cdot \Delta)$. ◀

In the context of graph reconstruction, the technique of random sampling and querying over the sample for the sake of set size estimation was first used by Mathieu & Zhou in [17] for a different purpose.

[1] When the notation $\tilde{O}(f(|X|))$ is used, the $\tilde{O}$ hides $\mathsf{polylog}(|X|)$ factors, not $\mathsf{polylog}(n)$.

5 Reconstruction of Spanning Trees

We next prove:

▶ **Lemma 16.** *Given a starshaped set X with center x, there is a randomized algorithm that uses $\tilde{O}(|X| \cdot \Delta^2)$ betweenness queries to learn a spanning tree of X.*

We first need the following technical results:

▶ **Claim 17.** *Let X be a starshaped set with center x, and let $v \in X$. Then $\mathcal{D}(v)$ is starshaped with center v.*

Proof. Let $u \in \mathcal{D}(v)$, let $\rho(u, v)$ be any shortest path between u and v, and let $w \in \rho(u, v)$. We then have $\mathrm{dist}(v, w) + \mathrm{dist}(w, u) = \mathrm{dist}(v, u)$. Since $u \in \mathcal{D}(v)$, we also have $\mathrm{dist}(x, v) + \mathrm{dist}(v, u) = \mathrm{dist}(x, u)$, and so combining these equations, we have

$$\mathrm{dist}(x, v) + \mathrm{dist}(v, w) = \mathrm{dist}(x, u) - \mathrm{dist}(w, u).$$

By the triangle inequality, we have $\mathrm{dist}(x, u) \leq \mathrm{dist}(x, w) + \mathrm{dist}(w, u)$, and so

$$\mathrm{dist}(x, v) + \mathrm{dist}(v, w) \leq \mathrm{dist}(x, w).$$

Again by the triangle inequality, we also have

$$\mathrm{dist}(x, v) + \mathrm{dist}(v, w) \geq \mathrm{dist}(x, w)$$

and so

$$\mathrm{dist}(x, v) + \mathrm{dist}(v, w) = \mathrm{dist}(x, w).$$

This implies that $w \in \mathcal{D}(v)$, and so $\mathcal{D}(v)$ is starshaped with center v. ◀

We omit the proofs of the following claims, as they are generally similar to the previous one.

▶ **Claim 18.** *Let X be a starshaped set with center x, and let $v \in X$. Then $X - \mathcal{D}(v)$ is starshaped with center x.*

▶ **Claim 19.** *Let X be a starshaped set with center x, and let $v \in X$. Then $(X - \mathcal{D}(v)) \cup \{v\}$ is starshaped with center x.*

These allow us to prove the following result:

▶ **Claim 20.** *Let X be a starshaped set with center x, and let $v \in X$. Let $\mathcal{T}_{(X-\mathcal{D}(v)) \cup \{v\}}$ be a spanning tree of the set $(X - \mathcal{D}(v)) \cup \{v\}$ with center x, and let $\mathcal{T}_{\mathcal{D}(v)}$ be a spanning tree of $\mathcal{D}(v)$ with center v. Then $\mathcal{T}_{(X-\mathcal{D}(v)) \cup \{v\}} \cup \mathcal{T}_{\mathcal{D}(v)}$ is a spanning tree of X.*

Proof. First, note that $\mathcal{T}_{(X-\mathcal{D}(v)) \cup \{v\}}$ and $\mathcal{T}_{\mathcal{D}(v)}$ have only the node v in common; therefore, their union is a tree.

Let $u \in X$. If $u \notin \mathcal{D}(v)$, then (by Claim 19) every shortest path between u and x is contained in $(X - \mathcal{D}(v)) \cup \{v\}$, and so $\mathcal{T}_{(X-\mathcal{D}(v)) \cup \{v\}}$ contains a shortest path between u and x.

Otherwise, suppose $u \in \mathcal{D}(v)$. Then $\mathcal{T}_{\mathcal{D}(v)}$ contains a shortest path between v and u, and $\mathcal{T}_{(X-\mathcal{D}(v)) \cup \{v\}}$ contains a shortest path between x and v. Since u is a descendant of v, we have $\mathrm{dist}(x, u) = \mathrm{dist}(x, v) + \mathrm{dist}(v, u)$. Therefore, the union of these two shortest paths is a shortest path between x and u.

We then have that $\mathcal{T}_{(X-\mathcal{D}(v)) \cup \{v\}} \cup \mathcal{T}_{\mathcal{D}(v)}$ is a tree that contains a shortest path between x and any $v \in X$, so it is a spanning tree of X. ◀

We can now return to Lemma 16.

Proof of Lemma 16. The algorithm is by recursive divide-and-conquer. The base case is when X consists of 2Δ or fewer nodes; in this case, it is possible to learn all edges of X in $O(\Delta^2)$ queries using Claim 12, and then find a spanning tree within these edges without any additional queries.

Repeat the following process. Use Lemma 15 to find a splitting node $s \in X$ in $\tilde{O}(|X| \cdot \Delta)$ queries. Next, query $\mathrm{anc}_x(s, v)$ for all $v \in X$ in order to determine the set $\mathcal{D}(s)$. Next, recursively compute a spanning tree of the sets $\mathcal{D}(s)$ and $(X - \mathcal{D}(s)) \cup \{s\}$ (these sets are starshaped by Claims 17 and 19), and then return the union of these two spanning trees. By Claim 20, this is a spanning tree of X.

The size of each set decreases by at least a factor of $(1 - \Theta(\frac{1}{\Delta}))$ at each round of the recursion, and so the recursion depth is $O(\log |X| \cdot \Delta)$. The number of top-level queries in each round is $\tilde{O}(|X| \cdot \Delta)$. This implies that $\tilde{O}(|X| \cdot \Delta^2)$ queries in total are used. ◀

▶ Remark. We note that the unweightedness of the hidden graph has not been exploited in any way, and therefore, Lemma 16 applies even if the hidden graph is weighted. More specifically, it is impossible for a betweenness query to learn the weight of a weighted edge, but one can find the edge set of a spanning tree using only $\tilde{O}(|X| \cdot \Delta^2)$ betweenness queries.

6 Reconstruction of Degree-Bounded Graphs

We closely follow the algorithm developed by Mathieu & Zhou in [17] for distance reconstruction. At a high level our algorithm is exactly the same as theirs (which is, in turn, based on techniques from [14] and [24]); our contribution is that we are able to use two of our previously established techniques to replace their distance queries with betweenness queries.

We prove:

▶ **Theorem 21.** *One can learn the edges of a hidden connected graph with maximum degree Δ in $\tilde{O}(n^{3/2} \cdot \Delta^4)$ betweenness queries.*

The following definition is critical to the proof:

▶ **Definition 22** (Cluster, [24]). Given a set of "centers" $A \subset V$ and a node $w \in V$, the *cluster* of w, denoted $C^A(w)$, is defined as $\{v \in V \mid \mathrm{dist}(w, v) < \mathrm{dist}(A, v)\}$.[2]

Mathieu & Zhou prove the following reduction (it is implicit in their Lemma 2):

▶ **Lemma 23** ([17]). *Let s be a parameter. Suppose there is an algorithm that learns the distance from a node v to each other node in a hidden graph, and that this algorithm uses $\leq f(n, \Delta)$ queries of some type. Then there is a randomized algorithm on this hidden graph that, with high probability, produces a set A of size $O(s \log n)$ such that $C^A(w) = O(n/s)$ for all $w \in V$. This randomized algorithm uses $\tilde{O}(f(n, \Delta) \cdot s)$ queries of the same type.*

Mathieu & Zhou use distance queries, and so trivially $f(n, \Delta) \leq n$. However, our Lemma 16 implies that for betweenness queries, we have $f(n, \Delta) = \tilde{O}(n \cdot \Delta^2)$. We can therefore find a set A as in Lemma 23 using $\tilde{O}(n \cdot \Delta^2 \cdot s)$ betweenness queries. The first step in the proof of Theorem 21 is to find A as in Lemma 23, and to find all pairwise distances in $A \times V$.

We next borrow some more methodology from Mathieu & Zhou.

[2] Where $\mathrm{dist}(A, v)$ is a shorthand for $\min_{a \in A} \mathrm{dist}(a, v)$.

▶ **Definition 24** ([17])**.** For each $a \in A$, define $\mathcal{C}(a)$ as

$$\{v \in V \mid \text{dist}(v, a) < \text{dist}(v, a') + 2 \text{ for all } a' \in A \setminus \{a\}\}$$

▶ **Claim 25** ([17])**.** *For all (hidden) edges $(u, v) \in G$, there exists an $a \in A$ such that $u, v \in \mathcal{C}(a)$.*

▶ **Claim 26** ([17])**.** $|\mathcal{C}(a)| = O(n/s \cdot \Delta^2)/$

The next claim is ours:

▶ **Claim 27.** *For all $a \in A$, the set $\mathcal{C}(a)$ is starshaped with center a.*

Proof. Let $u \in \mathcal{C}(a)$, let $\rho(a, u)$ be a shortest path between a and u, and let $w \in \rho(a, u)$. Suppose towards a contradiction that $w \notin \mathcal{C}(a)$; that is, there exists $a' \in A$ such that $\text{dist}(a, w) \geq \text{dist}(a', w) + 2$. We then have $\text{dist}(a, u) = \text{dist}(a, w) + \text{dist}(w, u)$, and by the triangle inequality, $\text{dist}(a', u) \leq \text{dist}(a', w) + \text{dist}(w, u)$. Combining these equations, we have

$$\begin{aligned}\text{dist}(a, u) - \text{dist}(a, w) &\geq \text{dist}(a', u) - \text{dist}(a', w)\\ &\geq \text{dist}(a', u) - (\text{dist}(a, w) - 2),\end{aligned}$$

and hence $\text{dist}(a, u) \geq \text{dist}(a', u) + 2$. Therefore $u \notin \mathcal{C}(a)$, which is a contradiction. We then have $w \in \mathcal{C}(a)$, and so $\mathcal{C}(a)$ is starshaped. ◀

We now return to Theorem 21.

Proof of Theorem 21. Compute the set A as in Lemma 23; by Lemma 16 this requires $\tilde{O}(n \cdot \Delta^2 \cdot s)$ betweenness queries. Once again use Lemma 16 to compute all pairwise distances in $A \times V$, and use this information to compute $\mathcal{C}(a)$ for all $a \in A$. Next, since each $\mathcal{C}(a)$ is starshaped (Claim 27), we can invoke Claim 12 to learn all edges in each set $\mathcal{C}(a)$. By Claim 26, the query complexity of this step is $|A| \cdot \tilde{O}(n^2/s^2 \cdot \Delta^4) = \tilde{O}(n^2/s \cdot \Delta^6)$. So the total query complexity of this process is $\tilde{O}(n \cdot \Delta^2 \cdot s) + \tilde{O}(n^2/s \cdot \Delta^6)$. The claim now follows by setting $s = n^{1/2}\Delta^2$. ◀

▶ Remark. A reasonable objection that can be made here is that we should model Δ as an unknown quantity, and so it is unfair to choose the value of s based on Δ. There is a way around this: one can make the assumption $\Delta = 2$ and run the algorithm as stated, aborting as soon as the runtime bound corresponding to $\Delta = 2$ is exceeded. If the bound is exceeded, then double the value of Δ and try again until the algorithm terminates successfully.

7 Reconstruction of Geodetic Graphs

Recall that a graph is *geodetic* if there is a unique shortest path between every pair of nodes. We next prove:

▶ **Theorem 28.** *One can learn the edges of a hidden geodetic connected graph with maximum degree Δ in $\tilde{O}(n \cdot \Delta^3)$ betweenness queries.*

This result was previously unknown, even for distance oracles.

We begin with a reduction of the problem. The input of our new problem is a starshaped set X with center x, a node $s \in X$, and a node $v \in (X - \mathcal{D}(s))$. Additionally, we are given the shortest path graph of X. The problem is to learn all edges with one endpoint at v and the other in $\mathcal{D}(s)$. We call this the *boundary edge problem.* Our reduction to this problem is as follows:

▶ **Lemma 29.** *Assume G is geodetic. Suppose there is a (possibly randomized) algorithm that solves the boundary edge problem using $f(|X|, \Delta)$ betweenness queries in the worst case. Then there is a randomized algorithm that, given a starshaped set X with center x, learns all edges in $G[X]$ using $\tilde{O}(|X| \cdot \Delta^3 \cdot f(|X|, \Delta))$ betweenness queries.*

Proof. We can learn the graph as follows. Use Lemma 16 to construct a spanning tree of X using $\tilde{O}(|X| \cdot \Delta^2)$ queries. Because the graph is geodetic, the spanning tree is also the shortest path graph of X. Next, find a splitting node s of X, using the algorithm outlined in Lemma 15. Solve the boundary edge problem for each node $v \in (X - \mathcal{D}(s))$; this requires $O(|X| \cdot f(|X|, \Delta))$ queries. We have now learned all edges with one endpoint in $\mathcal{D}(s)$ and the other endpoint in $X - \mathcal{D}(s)$; we still need to learn the edges with both endpoints in $\mathcal{D}(s)$ or both endpoints in $X - \mathcal{D}(s)$. We know that $\mathcal{D}(s)$ is starshaped with center s, and $X - \mathcal{D}(s)$ is starshaped with center x (by Claims 17 and 18), so this can be done recursively.

In each round of the recursion, we partition the node set into two subsets, each of which has at most a $1 - \Theta(1/\Delta)$ fraction as many nodes as in the previous round of the recursion. Therefore, the maximum recursion depth is $\tilde{O}(\Delta)$. The number of top-level queries made is $\tilde{O}(|X| \cdot \Delta^2 + |X| \cdot f(|X|, \Delta))$ and so the total query complexity of the algorithm is $\tilde{O}(|X| \cdot \Delta^3 + |X| \cdot \Delta \cdot f(|X|, \Delta))$. ◀

What remains is to place bounds on $f(|X|, \Delta)$, the query complexity of the boundary edge problem. To this end, the following claims are useful:

▶ **Claim 30.** *If the hidden graph is geodetic, then any edge (v, u) with $v \in (X - \mathcal{D}(s))$ and $u \in \mathcal{D}(s)$ are in the same layer in the set X.*

Proof. Let $u \in \mathcal{D}(s)$ be a neighbor of v. Let L_u^X be the layer of u, and let L_v^X be the layer of v. By the triangle inequality, we have that (1) $L_u^X = L_v^X + 1$, or (2) $L_u^X = L_v^X$, or (3) $L_u^X + 1 = L_v^X$. In fact, (3) is impossible because it implies that $v \in \mathcal{D}(s)$ and we have assumed $v \in (X - \mathcal{D}(s))$. Additionally, (1) is impossible because the hidden graph is geodetic. Specifically, since $u \in \mathcal{D}(s)$ there is a shortest path from x to u that passes through s; since $L_v^X + 1 = L_u^X$ there is another shortest path from x to u that passes through v (and therefore doesn't pass through s, since $v \notin \mathcal{D}(s)$); these shortest paths must be distinct. So (2) is the only possible case: we have $L_u^X = L_v^X$. ◀

▶ **Claim 31.** *If the hidden graph is geodetic, then for any edge (v, u) with $v \in (X - \mathcal{D}(s))$ and $u \in \mathcal{D}(s)$, we have $\operatorname{dist}(v, s) = \operatorname{dist}(u, s) + 1$.*

Proof. Let $L_u = L_v$ be the layer of u and v (these are the same by Claim 30), and let L_s be the layer of s. Since $u \in \mathcal{D}(s)$, we have $\operatorname{dist}(u, s) = L_u - L_s$. Therefore, it cannot be the case that $\operatorname{dist}(v, s) = \operatorname{dist}(u, s) - 1$: this would imply that $\operatorname{dist}(v, s) = L_v - L_s - 1$, which violates the triangle inequality. Additionally, it cannot be the case that $\operatorname{dist}(v, s) = \operatorname{dist}(u, s)$: this would imply that $\operatorname{dist}(v, s) = L_v - L_s$, and so $v \in \mathcal{D}(s)$, which we have assumed is not true. The only possibility that remains is that $\operatorname{dist}(v, s) = \operatorname{dist}(u, s) + 1$. ◀

▶ **Claim 32.** *If the hidden graph is geodetic, then each node $u \in (X - \mathcal{D}(s))$ has at most one incident edge in $\mathcal{D}(s)$.*

Proof. Let v be a neighbor of u in $\mathcal{D}(s)$. From Claim 31, we have that $\operatorname{dist}(u, s) = \operatorname{dist}(v, s) + 1$, and so u lies on a shortest path from v to s. If there are two distinct neighbors $u, u' \in \mathcal{D}(s)$ of u, then both of these neighbors lie on a shortest path from v to s, implying the existence of two distinct shortest paths from v to s. This violates our geodetic assumption. ◀

We can now prove:

▶ **Lemma 33.** *The boundary edge problem can be solved for hidden geodetic graphs in $\tilde{O}(\Delta)$ betweenness queries.*

Proof. The algorithm runs in two stages. First, we use $\tilde{O}(\Delta)$ betweenness queries to find a "candidate node" u, which is defined to be the node in $\mathcal{D}(s)$ furthest from x that lies on the shortest path between v and s. Note that if there exists an edge (v, u) with $u \in \mathcal{D}(s)$, then u will certainly be the candidate node by Claim 31. Therefore, once we have identified a candidate node u, it only remains to test whether or not the edge (u, v) is in the hidden graph, and we have detected the unique edge from v into $\mathcal{D}(s)$ or refuted its existence.

We can find the candidate node using our standard splitting node technique. Find a splitting node t of $\mathcal{D}(s)$ (we already have a shortest path graph of X, of which the shortest path graph of $\mathcal{D}(s)$ is a subgraph, so this can be computed offline and requires no queries). Query $\text{bet}(v, t, s)$. If true, then we recurse on the starshaped set $\mathcal{D}(t)$; if false, then we recurse on the starshaped set $\mathcal{D}(s) - \mathcal{D}(t)$, and repeat until only one node remains; this node is our candidate node. As usual, the recursion depth is $\tilde{O}(\Delta)$, and so we use this many betweenness queries.

Once we have our candidate node u, we can test for the existence of the edge (u, v) in a constant number of queries using Claim 11. ◀

Jointly, Lemmas 29 and 33 imply Theorem 28.

References

1 N. Alon and V. Asodi. Learning a hidden subgraph. *SIAM Journal of Discrete Math*, 18:697–712, 2005.

2 N. Alon, R. Beigel, S. Kasif, S. Riduch, and B. Sudakov. Learning a hidden matching. *SIAM Journal of Computing*, 33:487–501, 2004.

3 D. Angluin and J. Chen. Learning a hidden graph using $o(\log n)$ queries per edge. *COLT*, pages 210–223, 2004.

4 D. Angluin and J. Chen. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7:2215–2236, 2006.

5 Z. Beerliova, F. Eberhard, T. Erlebach, E. Hall, M. Hoffman, and L. S. Ram. Network discovery and verification. *WG*, pages 127–138, 2005.

6 M. Bouvel, V. Grebinski, and G. Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. *Graph-Theoretic Concepts in Computer Science*, pages 16–27, 2005.

7 R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: recent developments. *Statistical Science*, 19:499–517, 2004.

8 D. Chen, L. J. Guibas, J. Hershberger, and J. Sun. Road network reconstruction for organizing paths. *SODA*, pages 1309–1320, 2010.

9 L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006.

10 T. Erlebach, A. Hall, M. Hoffmann, and M. Mihal'ák. Network discovery and verification with distance queries. *Algorithms and Complexity*, pages 69–80, 2006.

11 V. Grebinski and G. Kucherov. Reconstring a hamiltonian cycle by querying the graph: Application to dna physical mapping. *Discrete Applied Mathematics 88*, pages 147–165, 1998.

12 V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica 28*, pages 104–124, 200.

13 J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.

14 S. Honiden, M. E. Houle, and C. Sommer. Balancing graph voronoi diagrams. *ISVD*, pages 183–191, 2009.

15 S. Kannan, C. Mathieu, and H. Zhou. Near-linear query complexity for graph inference. *Automata, Languages, and Programming*, 2015.

16 V. King, L. Zhang, and Y. Zhou. On the complexity of distance-based eviolutionary tree reconstruction. *SODA*, pages 444–453, 2003.

17 C. Mathieu and H. Zhou. Graph reconstruction via distance oracles. *ICALP*, pages 733–744, 2013.

18 H. Mazzawi. Optimally reconstructing weighted graphs using queries. *SODA*, pages 608–615, 2010.

19 S. Micali and Z. Allen-Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 2015.

20 M. Nei and J. Zhang. Evolutionary distance: Estimation. *Encyclopedia of Life Sciences*, 2005.

21 L. Reyzin and N. Srivastava. Learning and verifying graphs using queries with a focus on edge counting. *Proc. 18th Algorithmic Learning Theory*, pages 285–297, 2007.

22 L. Reyzin and N. Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Information processing letters*, 101(3):98–100, 2007.

23 F. Tarissan, M. Latapy, and C. Prieur. Efficient measurement of complex networks using link queries. *INFOCOM Workshops*, pages 254–259, 2009.

24 M. Thorup and U. Zwick. Compact routing schemes. *SPAA*, pages 1–10, 2001.

25 M. S. Waterman, T. F. Smith, M. Singh, and W. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.